פרויקט סיום - רשתות תקשורת

תוכן עניינים

מבוא

מה זה בכלל FTP?

שרת FTP זה שרת אשר אפשר להעלות אליו קבצים כגון תמונות, קבצי טקסט, קבצי מוזיקה ועוד. בנוסף אפשר להוריד ממנו בכל רגע נתון את הקבצים אשר העלנו אליו בעבר.

למה FTP בכלל שימושי?

ל-FTP יכולים להיות שימושים רבים:

1. העלאת קבצים לצורך גיבוי שלהם בעוד מקום.
2. העברת קבצים בין מחשבים בקלות ובמהירות.

איך FTP עובד?

כל מי שרוצה יוצר משתמש חדש אצל שרת ה FTP. השרת פותח אצלו תיקיה שבה הוא שומר את כל הקבצים אשר הלקוח מעלה אליו. השרת בכל רגע נתון יכול לתת ללקוח להוריד את הקבצים שאותו לקוח העלה בעבר.

שרת DHCP

DHCP, או Dynamic Host Configuration Protocol, הוא פרוטוקול לקוח ושרת המספק באופן אוטומטי כתובת פרוטוקול אינטרנט (IP) ומידע קשור אחר כגון מסכת רשת המשנה (subnet mask) ושער ברירת המחדל לנקודות קצה ברשת.

שרת ה-DHCP מקשיב בפורט 67. כאשר מסניף הודעת גילוי (Discovery), הוא משדר לפורט 68 הודעה משלו הכוללת כתובת IP ומידע נוסף. בשלב הבא, הלקוח מודיע לשרת שהוא אכן מוכן להשתמש ב-IP שקיבל ושולח הודעת request לפורט של השרת עם כתובת השרת על מנת שלא ייווצר קונפליקט בין שרתים. בהנחה שהשרת מקבל את הבקשה, הוא שולח הודעת אישור (acknowledge) ומודיע לנקודת הקצה שהיא זמינה.

שרת DNS

שרתי DNS מיועדים לענות על כל שאילתות מערכת שמות תחום (DNS). המטרה של שרת DNS היא לתרגם את מה שלקוח מקליד בדפדפן שלו למשהו שמחשב יכול להבין ולהשתמש בו כדי לתקשר עם אתר אינטרנט. במילים אחרות, מטרתו היא להמיר שם דומיין כגון www.example.com לכתובת ה-IP של שרת הדומיין.

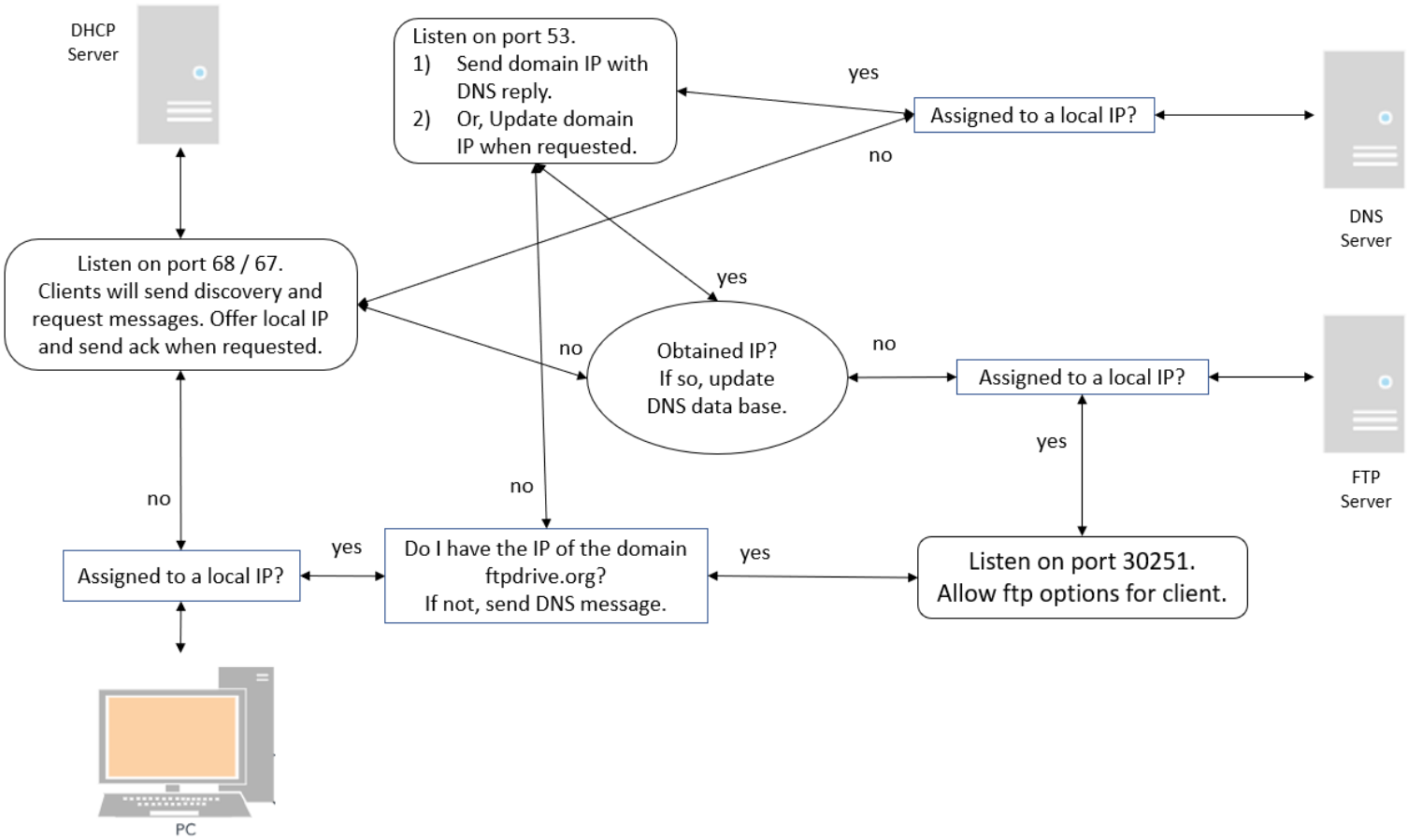
מהו שרת DNS ראשי?

שרת DNS ראשי הוא נקודת הקשר הראשונה של דפדפן שרוצה לדעת היכן למצוא אתר. השרת מכיל את מה שנקרא קובץ תחום (controlling zone file). קובץ זה מכיל את פרטי ה-DNS עבור הבקשה לתחום, כולל כתובת ה-IP שלו, כמו גם פרטי קשר של מנהל מערכת ורכיבים כמו Time to Live.

כיצד השרת שלנו מחפש את ה-IP של הדומיין?

תהליך הבירור לגבי תחום מסוים נעשה על ידי פנייה לשרת Top level המייצג את החלק האחרון בכתובת הדומיין. משם הוא יופנה לשרת הבא בהיררכיה לפי החלק הבא מימין לשמאל של הכתובת. לבסוף הוא יגיע לשרת שאחראי על הדומיין ושיוכל לספק את ה-IP המתבקש. תהליך בירור זה הוא תהליך איטרטיבי, ומתבצע בעיקר על ידי שרתים ששייכים לספקי אינטרנט. מחשבים אישיים בדרך כלל לא ידעו לבצע אותו, אלא רק לפנות לשרת של ספק האינטרנט על מנת שיבצע אותו עבורם.

דיאגרמת מצבים של התוכנית



איך מריצים את הפרויקט יחד עם DNS ו-DHCP

על מנת להריץ את שרת ה-DHCP שלנו נקליד:

```
nitay@ubuntu:~/Desktop/FinelProjectReshatot$ sudo python3 DHCP_Server.py
[sudo] password for nitay:
DHCP Server started.
DHCP Server IP: 10.0.3.1. DNS Server IP: 10.0.3.49.
```

השרת בנוי ככה שהוא לוקח את ה-IP של ממשק הרשת, מעלה את הספרה השלישית משמאל באחד והספרה הימנית ביותר הוא מגריל בין 2 ל254 עבור כל IP שהוא מעניק (הוא שומר את כתובות אותם העניק כבר ובודק שאלה שנותן אכן פנויות). בעת אתחול, השרת ייצר IP רנדומלי עבור שרת ה-DNS שלנו מבעוד מועד וה-IP שלו יהיה קבוע כאשר הספר האחרונה היא 1.

לאחר מכן נריץ את שרת ה-DNS בכך שנקליד:

```
nitay@ubuntu:~/Desktop/FinelProjectReshatot$ sudo python3 DNS_Server.py
[sudo] password for nitay:
Zone files loaded for domains:
    ftpdrive.org.
.
Sent 1 packets.
Received DHCP offer.
Server IP address:  10.0.3.1
Offered IP address:  10.0.3.20
Sending DHCP request for 10.0.3.49
.
Sent 1 packets.
Waiting for DHCP ACK
DHCP ACK received:  10.0.3.49
DNS server started on port 53 and IP 10.0.3.49
```

השרת קודם כל מדפיס את התחומים המקומיים (local domains) ששמורים כקבצי zone. לאחר מכן, שולח בקשה בפורט 67 עבור קבלת IP. ה-DHCP מסניף את הבקשה ושלוח לו הצעה ל-IP הכוללת מידע על כתובת ה-IP של שרת ה-DNS המיועדת לרשת.

השרת ישלח בקשה להשתמש ב-IP של ה-DNS אם הוא פנוי כדי להוות כשרת ה-DNS של הרשת (הכתובת במקרה זה היא 10.0.3.49).
מכיוון שהיא פנויה שרת ה-DHCP שולח אישור (Ack) והשרת מתחיל מקשיב בפורט 53 עבור בקשות DNS.

להלן צילום Wireshark:

No.	Time	Source	Destination	Protocol	Length	Info
58	10.958150935	0.0.0.0	255.255.255.255	DHCP	288	DHCP Discover - Transaction ID 0x0
59	10.959088230	10.0.2.2	10.0.2.16	DHCP	592	DHCP Offer - Transaction ID 0x0
73	13.080636281	10.0.3.1	10.0.3.20	DHCP	312	DHCP Offer - Transaction ID 0x0
75	14.140424601	0.0.0.0	255.255.255.255	DHCP	319	DHCP Request - Transaction ID 0x0
76	14.141684661	10.0.2.2	10.0.2.16	DHCP	592	DHCP ACK - Transaction ID 0x0
77	15.206499927	10.0.3.1	10.0.3.49	DHCP	312	DHCP ACK - Transaction ID 0x0

```
▸ Frame 58: 288 bytes on wire (2304 bits), 288 bytes captured (2304 bits) on interface any, id 0
▸ Linux cooked capture v1
▸ Internet Protocol Version 4, Src: 0.0.0.0, Dst: 255.255.255.255
▸ User Datagram Protocol, Src Port: 68, Dst Port: 67
▸ Dynamic Host Configuration Protocol (Discover)
```

השרת והלקוח ידעו לקחת רק את הצעות שרת ה-DHCP שלנו ולא של שרת ה-DHCP של הרשת אליה המחשב מחובר. לכן רואים שני הצעות ואישורים.

```
nitay@ubuntu:~/Desktop/FinelProjectReshatot$ sudo python3 DHCP_Server.py
[sudo] password for nitay:
DHCP Server started.
DHCP Server IP: 10.0.3.1. DNS Server IP: 10.0.3.49.
DHCP Discover received
.
Sent 1 packets.
DHCP Offer sent with IP: 10.0.3.20
DHCP Request received for IP: 10.0.3.49
.
Sent 1 packets.
DHCP ACK sent to 08:00:27:15:5a:a0 with domain IP: 10.0.3.49
```

מהצד של שרת ה-DHCP ניתן לראות הדפסות של בקשה ל-IP, מה הוא הציע, באיזה IP ההתקן מבקש להשתמש ואם נשלח אישור עם ה-IP הרצוי.

לאחר מכן ניתן להפעיל את שרת האפליקציה (FTP_Server או FTP_Server_TCP):

```
nitay@ubuntu:~/Desktop/FinelProjectReshatot$ sudo python3 FTP_Server.py
[sudo] password for nitay:
.
Sent 1 packets.
Received DHCP offer.
Server IP address:  10.0.3.1
Offered IP address:  10.0.3.175
DNS IP:  10.0.3.49
Sending DHCP request for 10.0.3.175
.
Sent 1 packets.
Waiting for DHCP ACK
DHCP ACK received:  10.0.3.175
start
```

באותו אופן הוא יקבל IP משרת ה-DHCP. לאחר שקיבל אישור, הוא ישלח הודעה לשרת ה-DNS שיעדכן את כתובת ה-IP של התחום (domain) הנקרא ftpdrive.org שמייצג את אפליקציית ה-FTP שלנו.

```
nitay@ubuntu:~/Desktop/FinelProjectReshatot$ sudo python3 DNS_Server.py
[sudo] password for nitay:
Zone files loaded for domains:
    ftpdrive.org.
.
Sent 1 packets.
Received DHCP offer.
Server IP address:  10.0.3.1
Offered IP address:  10.0.3.20
Sending DHCP request for 10.0.3.49
.
Sent 1 packets.
Waiting for DHCP ACK
DHCP ACK received:  10.0.3.49
DNS server started on port 53 and IP 10.0.3.49
Updated zone file for domain: ftpdrive.org
```

להלן תצלום של ה-Wireshark:

ip.addr == 10.0.3.0/24 or dhcp

No.	Time	Source	Destination	Protocol	Length	Info
58	10.958150935	0.0.0.0	255.255.255.255	DHCP	288	DHCP Discover - Transaction ID 0x0
59	10.959088230	10.0.2.2	10.0.2.16	DHCP	592	DHCP Offer - Transaction ID 0x0
73	13.080636281	10.0.3.1	10.0.3.20	DHCP	312	DHCP Offer - Transaction ID 0x0
75	14.140424601	0.0.0.0	255.255.255.255	DHCP	319	DHCP Request - Transaction ID 0x0
76	14.141684661	10.0.2.2	10.0.2.16	DHCP	592	DHCP ACK - Transaction ID 0x0
77	15.206499927	10.0.3.1	10.0.3.49	DHCP	312	DHCP ACK - Transaction ID 0x0
163	112.689159632	0.0.0.0	255.255.255.255	DHCP	288	DHCP Discover - Transaction ID 0x0
164	112.689895406	10.0.2.2	10.0.2.16	DHCP	592	DHCP Offer - Transaction ID 0x0
176	114.808117313	10.0.3.1	10.0.3.175	DHCP	312	DHCP Offer - Transaction ID 0x0
222	115.827960465	0.0.0.0	255.255.255.255	DHCP	319	DHCP Request - Transaction ID 0x0
223	115.829192100	10.0.2.2	10.0.2.16	DHCP	592	DHCP ACK - Transaction ID 0x0
224	116.864055035	10.0.3.1	10.0.3.175	DHCP	312	DHCP ACK - Transaction ID 0x0
225	116.891744039	10.0.3.175	10.0.3.49	DNS	57	Unknown operation (13) 0x646f[Malformed Packet]
226	116.992742140	10.0.3.175	10.0.3.49	DNS	56	Unknown operation (14) 0x6674[Malformed Packet]

```
▸ Frame 58: 288 bytes on wire (2304 bits), 288 bytes captured (2304 bits) on interface any, id 0
▸ Linux cooked capture v1
▸ Internet Protocol Version 4, Src: 0.0.0.0, Dst: 255.255.255.255
▸ User Datagram Protocol, Src Port: 68, Dst Port: 67
▸ Dynamic Host Configuration Protocol (Discover)
```

לבסוף אנו נפעיל את הלקוח באופן הבא (FTP_Client או FTP_Client_TCP):

```
nitay@ubuntu:~/Desktop/FinelProjectReshatot$ sudo python3 FTP_Client.py
[sudo] password for nitay:
.
Sent 1 packets.
Received DHCP offer.
Server IP address:  10.0.3.1
Offered IP address:  10.0.3.227
DNS IP:  10.0.3.49
Sending DHCP request for 10.0.3.227
.
Sent 1 packets.
Waiting for DHCP ACK
DHCP ACK received:  10.0.3.227
Received DNS reply. API IP:  10.0.3.175
Are you logged in or you need to register?
1 for register and 2 for log in:
```

בנוסף לקבלת IP הוא ישלח בקשת DNS מהשרת עבור הדומיין ftpdrive.org כאשר לאחר מכן הוא יחלץ את כתובת ה-IP של שרת האפליקציה מהתשובה של השרת. (ניתן להשתמש בפונקציית print_dns_reply של קובץ Net_Client על מנת להדפיס את כל המידע) בטרמינל של שרת ה-DNS ניתן לראות אל מי הוא החזיר תשובה לגבי הדומיין.

```
Sent 1 packets.
Waiting for DHCP ACK
DHCP ACK received:  10.0.3.49
DNS server started on port 53 and IP 10.0.3.49
Updated zone file for domain: ftpdrive.org
Sent response to ('10.0.3.227', 20599)
```

וב-Wireshark אנו רואים את חבילות ה-DNS בין הלקוח לשרת:

291	211.462776330	0.0.0.0	255.255.255.255	DHCP	288	DHCP Discover - Transaction ID 0x0
292	211.463406771	10.0.2.2	10.0.2.16	DHCP	592	DHCP Offer - Transaction ID 0x0
300	213.616475115	10.0.3.1	10.0.3.227	DHCP	312	DHCP Offer - Transaction ID 0x0
301	214.612421280	0.0.0.0	255.255.255.255	DHCP	319	DHCP Request - Transaction ID 0x0
302	214.613203916	10.0.2.2	10.0.2.16	DHCP	592	DHCP ACK - Transaction ID 0x0
304	215.671491717	10.0.3.1	10.0.3.227	DHCP	312	DHCP ACK - Transaction ID 0x0
305	215.707091143	10.0.3.227	10.0.3.49	DNS	74	Standard query 0x310d A ftpdrive.org
306	215.708419311	10.0.3.49	10.0.3.227	DNS	90	Standard query response 0x310d A ftpdrive.org A 10.0.3.175

(מצורף קובץ pcap בשם DHCP&DNSRUN)

הסבר כללי על האפליקציה עצמה עם RUDP

המטרה של הפרויקט היא להעביר את כל הקבצים בין שרת ללקוח(Download) או בין לקוח לשרת(Upload) לא משנה מה גודל הקובץ(הגודל המקסימלי שלנו זה MB64, אך אפשר בקלות להגדיל גודל זה על ידי שינוי גודל ה Buffer שבו השרת או הלקוח שומר את הקובץ המתקבל).

החלק של ה-ARQ ב-RUDP עובד אצלנו בשיטה הבאה: כל הודעה שנשלחת ה4 ביטים הראשונים שלה הם מספר ההודעה שנשלחה, כלומר ה4 ביטים האלה הם כמו ACK בTCP רק שאינם קשורים לאורך ההודעה ומתאפסים כל שליחה חדשה של קובץ. השרת בודק האם המספר הודעה

שנשלחה תואם למספר שלו הוא מצפה ואם לא השרת לא יסכים לקבל את ההודעה כלומר יהיה Timeout בין הלקוח לשרת. ברגע שיש Timeout כנראה שהחבילה שהשרת ציפה לה נאבדה ברשת ולכן הוא יחכה לאותה הודעה שהוא מצפה לה. בגלל ה-Timeout שהתקבל אצל הלקוח כי השרת לא שלח ack חזרה הלקוח ישלח שוב את ההודעה שנאבדה ברשת עד אשר השרת יחזיר לו תשובה. ניתן לראות את המימוש שהסברתי עכשיו בפונקציות הבאות אשר נמצאות בקוד של הלקוח והשרת:

```
FTP_Client.py   FTP_Server.py   pic.jpg
    return False


def try_send_pack(packeta):
    suc = False
    sock.sendto(packeta, server_address)
    while not suc:
        try:
            checkSeq, addressF = sock.recvfrom(packet_size)
            suc = True
        except TimeoutError:
            print("Packet was lost")
            sock.sendto(packeta, server_address)
            suc = False
    return checkSeq


def try_send_ack(packeta):
    suc = False
    sock.sendto(packeta, server_address)
    while not suc:
        try:
            checkSeq, addressF = sock.recvfrom(4)
            suc = True
        except TimeoutError:
            print("Packet was lost")
            sock.sendto(packeta, server_address)
            suc = False
    return int.from_bytes(checkSeq, 'big')
```

לאחר שהלקוח סיים לשלוח לשרת את הקובץ יש אצל השרת Timeout ואז השרת מבין שהלקוח סיים לשלוח את כל הקובץ, פותח אצלו קובץ חדש ומכניס את הביטים של הקובץ שקיבל מהלקוח לקובץ החדש בתוך השרת.
עכשיו השאלה המתבקשת היא איך אם חבילה נאבדת יש Timeout אצל הלקוח ואצל השרת אין Timeout? הגדרנו את זמן הTimeout אצל הלקוח להיות 0.1 שניות ואצל השרת להיות שנייה ולכן הסיכויים שיהיה Timeout אצל השרת לפני שסיימנו לשלוח את הקובץ יהיו מאוד נמוכים כי נשלח מהלקוח 10 הודעות לפני שיהיה Timeout אצל השרת.
נראה עכשיו צילומי מסך מה wireshark עם איבוד פאקטות ובלי איבוד פאקטות.

בלי איבוד פאקטות:

39	29.774139781	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
40	29.774252428	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
41	29.774300451	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
42	29.774340423	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
43	29.774389377	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
44	29.774492447	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
45	29.774540436	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
46	29.774583243	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
47	29.774633477	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
48	29.774769629	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
49	29.774818168	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
50	29.774861276	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
51	29.774910789	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
52	29.775028322	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
53	29.775167254	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
54	29.775219707	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
55	29.775280396	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
56	29.775386047	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
57	29.775436950	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
58	29.775484226	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
59	29.775534723	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
60	29.775638333	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
61	29.775687379	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
62	29.775729922	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
63	29.775780084	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
64	29.775881665	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
65	29.775928545	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
66	29.776001344	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
67	29.776064574	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
68	29.776170416	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
69	29.776224669	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
70	29.776266633	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
71	29.776315209	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
72	29.776416049	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
73	29.776462621	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
74	29.776504733	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4

ניתן לראות כאן שכל פעם אחרי שהלקוח שולח חלק מקובץ בגודל 64004 אז מתקבלת הודעה מהשרת בגודל 4 אשר מסמלת את שליחת הACK מהשרת ללקוח מה שאומר שהפאקטה הגיעה בשלום לשרת ולכן הלקוח ישלך את החלק הבא של הקובץ.

כך עשינו שלא יהיה איבוד פאקטות:

עכשיו נראה מה קורה כאשר הפעלנו איבוד פאקטות יחסית גדול של 20 אחוז:

```
vboxuser@Achiya:~$ sudo tc qdisc change dev lo root netem loss 20%
vboxuser@Achiya:~$
```

הווירשארק נראה כך:

89	29.850999204	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
90	29.851053100	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
91	29.851101150	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
92	29.952620052	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
93	29.952876243	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
94	29.953014842	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
95	29.953522581	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
96	29.953631729	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
97	29.953803678	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
98	29.953874804	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
99	29.953938431	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
100	29.954010528	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
101	29.954148294	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
102	29.954220284	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
103	29.954312844	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
104	29.954441305	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
105	29.954579019	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
106	29.954645131	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
107	29.954697327	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
108	29.954752790	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
109	29.954883031	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
110	29.954941483	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
111	29.954996256	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
112	29.955051680	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
113	30.056890690	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
114	30.159820821	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
115	30.363978067	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
116	30.364226784	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
117	30.364322043	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
118	30.364662714	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
119	30.364805528	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
120	30.364938066	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
121	30.365158144	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
122	30.466640646	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
123	30.466844902	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4
124	30.466991896	127.0.0.1	127.0.0.1	UDP	64046	20599 → 30251 Len=64004
125	30.467229022	127.0.0.1	127.0.0.1	UDP	46	30251 → 20599 Len=4

כאן אנחנו יכולים לראות שקרה לא מעט פעמים שהלקוח שלך את אותה הודעה כמה פעמים ברצף למשל בשורות 112-115 ורק לאחר שהוא קיבל ack מהשרת הוא שלח את החלק הבא של הקובץ. בצילום מסך זה מהווירשארק אפשר לראות ממש טוב איך החלק של הARQ של הRUDP שלנו עובד כמו שצריך וכמו שהסברנו.

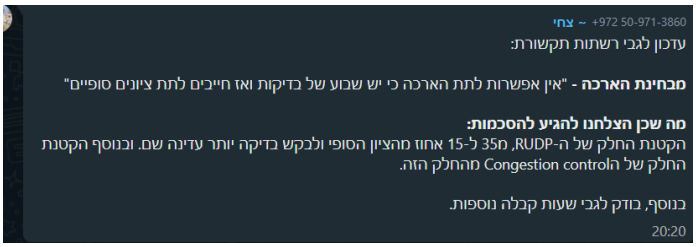
<u>בעיות של Latency:</u> אם ה latency הוא יותר משנייה שזה זמן הTimeout של השרת אז כנראה שמשהו לא כל כך טוב עם הרשת והקובץ ישמר עד לאן שהשליחה התקבלה בתוך השרת. אם ה latency הוא פחות משנייה אז הלקוח פשוט ישלח עוד פעם ועוד פעם את החלק של הקובץ עד שהשרת ישלח לו הודעת ack חזרה וכך המערכת תתגבר על latency של פחות משנייה.

<u>Flow Control:</u>

השתמשנו ב-Flow Control פשוט מפאת חוסר הזמן שנתקלנו בו.
ה-Flow Control בו בחרנו הוא stop and wait. עד שהשרת לא פנוי לקבל הודעה, כלומר עד שהוא מגיע לשורת ה-recvfrom בקוד אז הלקוח לא ישלח את ההודעה וכך בוודאות לא ייווצר מצב שה-buffer של השרת יתמלא יתר על המידה כי השרת מטפל כל פעם בהודעה אחת ואז חוזר לקבל הודעה חדשה לאחר שסיים לטפל בהודעה הראשונה.

<u>Cognition Control:</u>

פה גם המימוש מאוד פשוט מפאת חוסר הזמן שלנו ובנוסף נשלחה הודעה שהורידו את החלק הזה מהבדיקה של המטלה כמו שאפשר לראות בתמונה המצורפת כאן:



בגלל שכל פעם אנו שולחים ומקבלים רק הודעה אחת אז לא צריך איזה שהוא אלגוריתם של Cognition Control כי אנחנו לא מגדילים את חלון השליחה אלא תמיד משאירים אותו על גודל 1.

הבדלים בין הרצת ה-TCP לבין הרצת ה-RUDP עם/בלי איבוד פאקטות

פרוטוקול RUDP

בהרצה רגילה אנו נפעיל את שרתי ה-DHCP, DNS קודם ובשלב הבא אם נרצה להריץ את הלקוח ושרת ה-FTP באמצעות RUDP נקליד:

```
nitay@ubuntu:~/Desktop/FinelProjectReshatot$ sudo python3 FTP_Server.py
[sudo] password for nitay:
```

ואחריו הלקוח:

```
nitay@ubuntu:~/Desktop/FinelProjectReshatot$ sudo python3 FTP_Client.py
[sudo] password for nitay:
```

לאחר שהלקוח יקבל את ה-IP של שרת ה-FTP משרת ה-DNS ניתן לבקש משרת ה-FTP שלוש אפשרויות: להעלות קובץ, להוריד, או לצפות ברשימת הקבצים שבשרת המסודרים לפי שם:

```
1. Upload a file to the FTP server
2. Download a file from the FTP server
3. View all the files you can download
Any other character to exit
```

נשלח בקשה להעלות תמונה לשרת וב-Wireshark ניתן לראות חבילות UDP בין הלקוח לשרת:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.3.6	10.0.3.84	UDP	64	20599 → 30251 Len=20
2	0.000443654	10.0.3.84	10.0.3.6	UDP	48	30251 → 20599 Len=4
3	0.041465433	10.0.3.6	10.0.3.84	UDP	64048	20599 → 30251 Len=64004
4	0.042194112	10.0.3.84	10.0.3.6	UDP	48	30251 → 20599 Len=4
5	0.043294522	10.0.3.6	10.0.3.84	UDP	64048	20599 → 30251 Len=64004
6	0.043373263	10.0.3.84	10.0.3.6	UDP	48	30251 → 20599 Len=4
7	0.043549241	10.0.3.6	10.0.3.84	UDP	64048	20599 → 30251 Len=64004
8	0.043620927	10.0.3.84	10.0.3.6	UDP	48	30251 → 20599 Len=4
9	0.043834466	10.0.3.6	10.0.3.84	UDP	64048	20599 → 30251 Len=64004
10	0.043936463	10.0.3.84	10.0.3.6	UDP	48	30251 → 20599 Len=4
11	0.044115566	10.0.3.6	10.0.3.84	UDP	64048	20599 → 30251 Len=64004
12	0.044185510	10.0.3.84	10.0.3.6	UDP	48	30251 → 20599 Len=4
13	0.044339023	10.0.3.6	10.0.3.84	UDP	64048	20599 → 30251 Len=64004
14	0.044399311	10.0.3.84	10.0.3.6	UDP	48	30251 → 20599 Len=4
15	0.044553347	10.0.3.6	10.0.3.84	UDP	64048	20599 → 30251 Len=64004
16	0.044624749	10.0.3.84	10.0.3.6	UDP	48	30251 → 20599 Len=4
17	0.044827099	10.0.3.6	10.0.3.84	UDP	64048	20599 → 30251 Len=64004
18	0.044902762	10.0.3.84	10.0.3.6	UDP	48	30251 → 20599 Len=4
19	0.045070006	10.0.3.6	10.0.3.84	UDP	64048	20599 → 30251 Len=64004
20	0.045137261	10.0.3.84	10.0.3.6	UDP	48	30251 → 20599 Len=4
21	0.045287374	10.0.3.6	10.0.3.84	UDP	64048	20599 → 30251 Len=64004
22	0.045349614	10.0.3.84	10.0.3.6	UDP	48	30251 → 20599 Len=4
23	0.045515221	10.0.3.6	10.0.3.84	UDP	64048	20599 → 30251 Len=64004
24	0.045583864	10.0.3.84	10.0.3.6	UDP	48	30251 → 20599 Len=4
25	0.045780469	10.0.3.6	10.0.3.84	UDP	64048	20599 → 30251 Len=64004
26	0.045855769	10.0.3.84	10.0.3.6	UDP	48	30251 → 20599 Len=4
27	0.046059166	10.0.3.6	10.0.3.84	UDP	64048	20599 → 30251 Len=64004
28	0.046323501	10.0.3.84	10.0.3.6	UDP	48	30251 → 20599 Len=4
29	0.048315473	10.0.3.6	10.0.3.84	UDP	64048	20599 → 30251 Len=64004
30	0.049107532	10.0.3.84	10.0.3.6	UDP	48	30251 → 20599 Len=4
31	0.051251867	10.0.3.6	10.0.3.84	UDP	64048	20599 → 30251 Len=64004

```
▸ Frame 1: 64 bytes on wire (512 bits), 64 bytes captured (512 bits) on interface any, id 0
▸ Linux cooked capture v1
▸ Internet Protocol Version 4, Src: 10.0.3.6, Dst: 10.0.3.84
▸ User Datagram Protocol, Src Port: 20599, Dst Port: 30251
▸ Data (20 bytes)
```

(ראה קובץ pcap בשם rudpwithoutloss)

בהרצה שונה, אנו נפעיל איבוד פאקטות כפי שעשינו במטלה 3 ונבחין כי מכיוון שפרוטוקול RUDP אכן אמין הוא מצליח לעקוב אחר החבילות שנאבדו לפי ה-seq שאינם הופיעו בסדר כרונולוגי ולבסוף להעלות קובץ לשרת.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.3.47	10.0.3.92	UDP	64	20599 → 30251 Len=20
2	0.000222813	10.0.3.92	10.0.3.47	UDP	48	30251 → 20599 Len=4
3	0.205999855	10.0.3.47	10.0.3.92	UDP	64048	20599 → 30251 Len=64004
4	0.206124392	10.0.3.92	10.0.3.47	UDP	48	30251 → 20599 Len=4
5	0.206234214	10.0.3.47	10.0.3.92	UDP	64048	20599 → 30251 Len=64004
6	0.206293876	10.0.3.92	10.0.3.47	UDP	48	30251 → 20599 Len=4
7	0.206369048	10.0.3.47	10.0.3.92	UDP	64048	20599 → 30251 Len=64004
8	0.206423699	10.0.3.92	10.0.3.47	UDP	48	30251 → 20599 Len=4
9	0.206650465	10.0.3.47	10.0.3.92	UDP	64048	20599 → 30251 Len=64004
10	0.206710293	10.0.3.92	10.0.3.47	UDP	48	30251 → 20599 Len=4
11	0.206779178	10.0.3.47	10.0.3.92	UDP	64048	20599 → 30251 Len=64004
12	0.206835577	10.0.3.92	10.0.3.47	UDP	48	30251 → 20599 Len=4
13	0.307385821	10.0.3.47	10.0.3.92	UDP	64048	20599 → 30251 Len=64004
14	0.407762660	10.0.3.47	10.0.3.92	UDP	64048	20599 → 30251 Len=64004
15	0.411043480	10.0.3.92	10.0.3.47	UDP	48	30251 → 20599 Len=4
16	0.411173951	10.0.3.47	10.0.3.92	UDP	64048	20599 → 30251 Len=64004
17	0.511632691	10.0.3.47	10.0.3.92	UDP	64048	20599 → 30251 Len=64004
18	0.614007460	10.0.3.47	10.0.3.92	UDP	64048	20599 → 30251 Len=64004
19	0.614134350	10.0.3.92	10.0.3.47	UDP	48	30251 → 20599 Len=4
20	0.614270921	10.0.3.47	10.0.3.92	UDP	64048	20599 → 30251 Len=64004
21	0.614334831	10.0.3.92	10.0.3.47	UDP	48	30251 → 20599 Len=4
22	0.614546242	10.0.3.47	10.0.3.92	UDP	64048	20599 → 30251 Len=64004
23	0.614610545	10.0.3.92	10.0.3.47	UDP	48	30251 → 20599 Len=4
24	0.715255884	10.0.3.47	10.0.3.92	UDP	64048	20599 → 30251 Len=64004
25	0.715439363	10.0.3.92	10.0.3.47	UDP	48	30251 → 20599 Len=4
26	0.715609690	10.0.3.47	10.0.3.92	UDP	64048	20599 → 30251 Len=64004
27	0.715720359	10.0.3.92	10.0.3.47	UDP	48	30251 → 20599 Len=4
28	0.715827371	10.0.3.47	10.0.3.92	UDP	64048	20599 → 30251 Len=64004
29	0.715917572	10.0.3.92	10.0.3.47	UDP	48	30251 → 20599 Len=4
30	0.716165782	10.0.3.47	10.0.3.92	UDP	64048	20599 → 30251 Len=64004
31	0.716279005	10.0.3.92	10.0.3.47	UDP	48	30251 → 20599 Len=4
32	0.716390243	10.0.3.47	10.0.3.92	UDP	64048	20599 → 30251 Len=64004
33	0.716546611	10.0.3.92	10.0.3.47	UDP	48	30251 → 20599 Len=4
34	0.716735317	10.0.3.47	10.0.3.92	UDP	64048	20599 → 30251 Len=64004

(ראה קובץ pcap בשם rudpwithloss20)

פרוטוקול TCP

אם נרצה להריץ את הלקוח והשרת עם TCP נקליד כך:

```
nitay@ubuntu:~/Desktop/FinelProjectReshatot$ sudo python3 FTP_Server_TCP.py
[sudo] password for nitay:
```

ואז את הלקוח:

```
nitay@ubuntu:~/Desktop/FinelProjectReshatot$ sudo python3 FTP_Client_TCP.py
[sudo] password for nitay:
```

לאחר שהלקוח יקבל את ה-IP של שרת ה-FTP משרת ה-DNS ניתן לבקש משרת ה-FTP שש אפשרויות: להעלות קובץ, להוריד, לצפות ברשימת הקבצים שבשרת המסודרים לפי שם, להעלות קובץ לתיקייה של חבר בשרת, או לרוקן את התיקייה שלנו:

```
Choose an action:
1. Upload a file to the FTP server
2. Download a file from the FTP server
3. View all the files you can download
4. send a file to a friend
5. Empty Directory
6. To exit.
```

נשים לב שמתקיים לחיצת ידיים משולשת כנהוג בפרוטוקול TCP:

tcp.port == 20599 or tcp.port == 30251

No.	Time	Source	Destination	Protocol	Length	Info
314	92.820518827	10.0.3.213	10.0.3.185	TCP	76	20599 → 30251 [SYN] Se
315	92.820546190	10.0.3.185	10.0.3.213	TCP	76	30251 → 20599 [SYN, AC
316	92.820564870	10.0.3.213	10.0.3.185	TCP	68	20599 → 30251 [ACK] Se

אם נרצה, ניתן לבקש לעצור באמצע שליחת הקובץ או הורדת הקובץ:

```
Server answer: ACK
File transfer started
Uploading pic.jpg...
File size: 5245329 bytes
Do you want to stop middle packet upload?
no: 0, yes: any.
```

נבקש להעלות קובץ תמונה לשרת ונעקוב אחר החבילות ב-Wireshark:

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.3.213	10.0.3.185	TCP	76	20599 → 30251 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=3357966285 TSecr=0 WS=128
2	0.000027363	10.0.3.185	10.0.3.213	TCP	76	30251 → 20599 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3462191089 TSecr=335.
3	0.000046043	10.0.3.213	10.0.3.185	TCP	68	20599 → 30251 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3357966285 TSecr=3462191089
4	51.535816656	10.0.3.213	10.0.3.185	TCP	83	20599 → 30251 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=15 TSval=3358017821 TSecr=3462191089
5	51.535840502	10.0.3.185	10.0.3.213	TCP	68	30251 → 20599 [ACK] Seq=1 Ack=16 Win=65536 Len=0 TSval=3462242625 TSecr=3358017821
6	51.536015998	10.0.3.185	10.0.3.213	TCP	71	30251 → 20599 [PSH, ACK] Seq=1 Ack=16 Win=65536 Len=3 TSval=3462242625 TSecr=3358017821
7	51.536031238	10.0.3.213	10.0.3.185	TCP	68	20599 → 30251 [ACK] Seq=16 Ack=4 Win=65536 Len=0 TSval=3358017821 TSecr=3462242625
8	67.772422053	10.0.3.213	10.0.3.185	TCP	32836	20599 → 30251 [ACK] Seq=16 Ack=4 Win=65536 Len=32768 TSval=3358034057 TSecr=3462242625
9	67.772450874	10.0.3.213	10.0.3.185	TCP	31300	20599 → 30251 [PSH, ACK] Seq=32784 Ack=4 Win=65536 Len=31232 TSval=3358034057 TSecr=3462242625
10	67.772534924	10.0.3.185	10.0.3.213	TCP	68	30251 → 20599 [ACK] Seq=4 Ack=64016 Win=65536 Len=0 TSval=3462258861 TSecr=3358034057
11	67.772555545	10.0.3.213	10.0.3.185	TCP	32836	20599 → 30251 [ACK] Seq=64016 Ack=4 Win=65536 Len=32768 TSval=3358034057 TSecr=3462258861
12	67.772565935	10.0.3.213	10.0.3.185	TCP	31300	20599 → 30251 [PSH, ACK] Seq=96784 Ack=4 Win=65536 Len=31232 TSval=3358034057 TSecr=3462258861
13	67.772720726	10.0.3.185	10.0.3.213	TCP	68	30251 → 20599 [ACK] Seq=4 Ack=128016 Win=65536 Len=0 TSval=3462258862 TSecr=3358034057
14	67.772741615	10.0.3.213	10.0.3.185	TCP	32836	20599 → 30251 [ACK] Seq=128016 Ack=4 Win=65536 Len=32768 TSval=3358034058 TSecr=3462258862
15	67.772752240	10.0.3.213	10.0.3.185	TCP	32836	[TCP Window Full] 20599 → 30251 [ACK] Seq=160784 Ack=4 Win=65536 Len=32768 TSval=3358034058 TSecr=346.
16	67.772756709	10.0.3.185	10.0.3.213	TCP	68	30251 → 20599 [ACK] Seq=4 Ack=193552 Win=327424 Len=0 TSval=3462258862 TSecr=3358034058
17	67.772768529	10.0.3.213	10.0.3.185	TCP	32836	20599 → 30251 [PSH, ACK] Seq=193552 Ack=4 Win=65536 Len=32768 TSval=3358034058 TSecr=3462258862
18	67.772779070	10.0.3.213	10.0.3.185	TCP	32836	20599 → 30251 [ACK] Seq=226320 Ack=4 Win=65536 Len=32768 TSval=3358034058 TSecr=3462258862
19	67.772783432	10.0.3.185	10.0.3.213	TCP	68	30251 → 20599 [ACK] Seq=4 Ack=259088 Win=589440 Len=0 TSval=3462258862 TSecr=3358034058
20	67.772791876	10.0.3.213	10.0.3.185	TCP	32836	20599 → 30251 [PSH, ACK] Seq=259088 Ack=4 Win=65536 Len=32768 TSval=3358034058 TSecr=3462258862
21	67.772801064	10.0.3.213	10.0.3.185	TCP	32836	20599 → 30251 [ACK] Seq=291856 Ack=4 Win=65536 Len=32768 TSval=3358034058 TSecr=3462258862
22	67.772805073	10.0.3.185	10.0.3.213	TCP	68	30251 → 20599 [ACK] Seq=4 Ack=324624 Win=851328 Len=0 TSval=3462258862 TSecr=3358034058
23	67.772813746	10.0.3.213	10.0.3.185	TCP	32836	20599 → 30251 [PSH, ACK] Seq=324624 Ack=4 Win=65536 Len=32768 TSval=3358034058 TSecr=3462258862
24	67.772875017	10.0.3.213	10.0.3.185	TCP	65551	20599 → 30251 [ACK] Seq=357392 Ack=4 Win=65536 Len=65483 TSval=3358034058 TSecr=3462258862
25	67.772880518	10.0.3.185	10.0.3.213	TCP	68	30251 → 20599 [ACK] Seq=4 Ack=422875 Win=1113216 Len=0 TSval=3462258862 TSecr=3358034058
26	67.772896894	10.0.3.213	10.0.3.185	TCP	25209	20599 → 30251 [PSH, ACK] Seq=422875 Ack=4 Win=65536 Len=25141 TSval=3358034058 TSecr=3462258862
27	67.773040743	10.0.3.213	10.0.3.185	TCP	65551	20599 → 30251 [ACK] Seq=448016 Ack=4 Win=65536 Len=65483 TSval=3358034058 TSecr=3462258862
28	67.773047947	10.0.3.185	10.0.3.213	TCP	68	30251 → 20599 [ACK] Seq=4 Ack=513499 Win=1375232 Len=0 TSval=3462258862 TSecr=3358034058
29	67.773074533	10.0.3.213	10.0.3.185	TCP	62585	20599 → 30251 [PSH, ACK] Seq=513499 Ack=4 Win=65536 Len=62517 TSval=3358034058 TSecr=3462258862
30	67.773151871	10.0.3.213	10.0.3.185	TCP	65551	20599 → 30251 [ACK] Seq=576016 Ack=4 Win=65536 Len=65483 TSval=3358034058 TSecr=3462258862
31	67.773157516	10.0.3.185	10.0.3.213	TCP	68	30251 → 20599 [ACK] Seq=4 Ack=641499 Win=1637120 Len=0 TSval=3462258862 TSecr=3358034058

▾ Transmission Control Protocol, Src Port: 20599, Dst Port: 30251, Seq: 32784, Ack: 4, Len: 31232

(ראה קובץ pcap בשם tcpwithoutloss)

בהרצה שונה, אנו נפעיל איבוד פאקטות כפי שעשינו במטלה 3 ונבחין כי מכיוון שפרוטוקל TCP אכן אמין הוא מצליח לעקוב אחר החבילות שנאבדו ולבסוף להעלות קובץ לשרת.

No.	Time	Source	Destination	Protocol	Length	Info
31	16.286716934	10.0.3.4	10.0.3.131	TCP	32836	20599 → 30251 [PSH, ACK] Seq=457232 Ack=4 Win=65536 Len=32768 TSval=2191953186 TSecr=3856163597
32	16.286733384	10.0.3.131	10.0.3.4	TCP	68	30251 → 20599 [ACK] Seq=4 Ack=490000 Win=1637120 Len=0 TSval=3856163597 TSecr=2191953186
33	16.286869520	10.0.3.4	10.0.3.131	TCP	32836	20599 → 30251 [ACK] Seq=490000 Ack=4 Win=65536 Len=32768 TSval=2191953186 TSecr=3856163597
34	16.286896877	10.0.3.131	10.0.3.4	TCP	68	30251 → 20599 [ACK] Seq=4 Ack=522768 Win=1768064 Len=0 TSval=3856163598 TSecr=2191953186
35	16.286930739	10.0.3.4	10.0.3.131	TCP	32836	20599 → 30251 [PSH, ACK] Seq=522768 Ack=4 Win=65536 Len=32768 TSval=2191953186 TSecr=3856163597
36	16.286947870	10.0.3.131	10.0.3.4	TCP	68	30251 → 20599 [ACK] Seq=4 Ack=555536 Win=1899008 Len=0 TSval=3856163598 TSecr=2191953186
37	16.286984857	10.0.3.4	10.0.3.131	TCP	32836	20599 → 30251 [ACK] Seq=555536 Ack=4 Win=65536 Len=32768 TSval=2191953187 TSecr=3856163598
38	16.346955152	10.0.3.4	10.0.3.131	TCP	32836	[TCP Previous segment not captured] 20599 → 30251 [ACK] Seq=621072 Ack=4 Win=65536 Len=32768 TSval=21…
39	16.346976316	10.0.3.131	10.0.3.4	TCP	80	30251 → 20599 [ACK] Seq=4 Ack=588304 Win=2161024 Len=0 TSval=3856163658 TSecr=2191953187 SLE=621072 S…
40	16.347025861	10.0.3.4	10.0.3.131	TCP	32836	20599 → 30251 [PSH, ACK] Seq=653840 Ack=4 Win=65536 Len=32768 TSval=2191953247 TSecr=3856163658
41	16.347037672	10.0.3.4	10.0.3.131	TCP	32836	20599 → 30251 [ACK] Seq=686608 Ack=4 Win=65536 Len=32768 TSval=2191953247 TSecr=3856163658
42	16.347048050	10.0.3.4	10.0.3.131	TCP	32836	20599 → 30251 [PSH, ACK] Seq=719376 Ack=4 Win=65536 Len=32768 TSval=2191953247 TSecr=3856163658
43	16.347054956	10.0.3.131	10.0.3.4	TCP	80	[TCP Window Update] 30251 → 20599 [ACK] Seq=4 Ack=588304 Win=2291968 Len=0 TSval=3856163658 TSecr=219…
44	16.347079546	10.0.3.131	10.0.3.4	TCP	80	[TCP Window Update] 30251 → 20599 [ACK] Seq=4 Ack=588304 Win=2372864 Len=0 TSval=3856163658 TSecr=219…
45	16.347093646	10.0.3.4	10.0.3.131	TCP	32836	[TCP Out-Of-Order] 20599 → 30251 [PSH, ACK] Seq=588304 Ack=4 Win=65536 Len=32768 TSval=2191953247 TSe…
46	16.347117705	10.0.3.131	10.0.3.4	TCP	68	30251 → 20599 [ACK] Seq=4 Ack=752144 Win=2340736 Len=0 TSval=3856163658 TSecr=2191953247
47	16.399082742	10.0.3.4	10.0.3.131	TCP	32836	[TCP Previous segment not captured] 20599 → 30251 [ACK] Seq=817680 Ack=4 Win=65536 Len=32768 TSval=21…
48	16.399099296	10.0.3.131	10.0.3.4	TCP	80	[TCP Window Update] 30251 → 20599 [ACK] Seq=4 Ack=752144 Win=2406912 Len=0 TSval=3856163710 TSecr=219…
49	16.656083214	10.0.3.4	10.0.3.131	TCP	32836	[TCP Retransmission] 20599 → 30251 [ACK] Seq=752144 Ack=4 Win=65536 Len=32768 TSval=2191953556 TSecr=…
50	16.656107130	10.0.3.131	10.0.3.4	TCP	80	30251 → 20599 [ACK] Seq=4 Ack=784912 Win=2389888 Len=0 TSval=3856163967 TSecr=2191953556 SLE=817680 S…
51	16.656135138	10.0.3.4	10.0.3.131	TCP	32836	[TCP Retransmission] 20599 → 30251 [PSH, ACK] Seq=784912 Ack=4 Win=65536 Len=32768 TSval=2191953556 T…
52	16.656145118	10.0.3.4	10.0.3.131	TCP	32836	20599 → 30251 [PSH, ACK] Seq=850448 Ack=4 Win=65536 Len=32768 TSval=2191953556 TSecr=3856163967
53	16.656149324	10.0.3.131	10.0.3.4	TCP	68	30251 → 20599 [ACK] Seq=4 Ack=850448 Win=2374144 Len=0 TSval=3856163967 TSecr=2191953556
54	16.694184192	10.0.3.4	10.0.3.131	TCP	32836	[TCP Previous segment not captured] 20599 → 30251 [PSH, ACK] Seq=915984 Ack=4 Win=65536 Len=32768 TSv…
55	16.694197469	10.0.3.131	10.0.3.4	TCP	80	30251 → 20599 [ACK] Seq=4 Ack=883216 Win=2406912 Len=0 TSval=3856164005 TSecr=2191953556 SLE=915984 S…
56	16.694216874	10.0.3.4	10.0.3.131	TCP	32836	20599 → 30251 [ACK] Seq=948752 Ack=4 Win=65536 Len=32768 TSval=2191953594 TSecr=3856164005
57	16.694220706	10.0.3.131	10.0.3.4	TCP	80	[TCP Dup ACK 55#1] 30251 → 20599 [ACK] Seq=4 Ack=883216 Win=2406912 Len=0 TSval=3856164005 TSecr=2191…
58	16.694235232	10.0.3.4	10.0.3.131	TCP	32836	[TCP Out-Of-Order] 20599 → 30251 [ACK] Seq=883216 Ack=4 Win=65536 Len=32768 TSval=2191953594 TSecr=38…
59	16.694245359	10.0.3.131	10.0.3.4	TCP	68	30251 → 20599 [ACK] Seq=4 Ack=981520 Win=2373504 Len=0 TSval=3856164005 TSecr=2191953594
60	16.694266391	10.0.3.4	10.0.3.131	TCP	32836	20599 → 30251 [PSH, ACK] Seq=981520 Ack=4 Win=65536 Len=32768 TSval=2191953594 TSecr=3856164005
61	16.694270438	10.0.3.4	10.0.3.131	TCP	32836	20599 → 30251 [ACK] Seq=1014288 Ack=4 Win=65536 Len=32768 TSval=2191953594 TSecr=3856164005

(ראה קובץ pcap בשם tcpwithloss20)

פתרונות לשאלות הפרויקט

1) ארבע הבדלים עיקריים בין פרוטוקול TCP ל-QUIC

לחיצת יד: TCP משתמש בלחיצת יד משולשת כדי ליצור חיבור לפני שליחת המידע, ובנוסף לכך, נדרש לנהל אבטחת שכבת התעבורה (TLS) בין הצדדים. לעומת זאת, QUIC בנוי על גבי UDP כך שהוא משתמש בתהליך לחיצת יד מהיר יותר הדורש רק חבילה אחת הלוך ושוב, כולל TLS. זה יכול לגרום לזמני יצירת חיבור מהירים יותר והשהייה נמוכה יותר.

ריבוי: מכיוון ש-TCP משתמש רק בחיבור אחד עבור כל העברת נתונים, כל חיבור יכול לשאת נתונים רק עבור יישום אחד בכל פעם. בניגוד אליו, QUIC מאפשר ריבוי, הנותן ליישומים שונים לשתף חיבור יחיד. כתוצאה מכך, ניתן להשתמש במשאבי הרשת בצורה יעילה יותר, והשהייה פחותה.

אמינות ויעילות: TCP הוא פרוטוקול אמין המבטיח שהנתונים מועברים בסדר וללא שגיאות. הוא משתמש באישורים ובשידורים חוזרים כדי להבטיח מסירה אמינה. QUIC הוא גם פרוטוקול אמין, אך הוא מסתמך על טכניקה הנקראת תיקון שגיאות קדימה (FEC).
האסטרטגיה היא להוסיף באופן שיטתי נתונים לתיקון שגיאות כדי להיות מסוגל לשחזר את הנתונים המשודרים שאבדו במעבר.
לפיכך, FEC נמנע מהשהייה גבוהה ב-Round-Trip Time (RTT) באמצעות הגדלת כמות הנתונים שיועברו, ובמידת הצורך, חישוב הנתונים האבודים.
כך המקבל עדיין יכול לשחזר את הנתונים המקוריים מבלי לדרוש שידורים חוזרים שייקחו זמן רב.

הצפנה: בעוד ש-TCP תומך בהצפנה באמצעות שימוש בפרוטוקולים כמו TLS (לא חלק מרכזי בפרוטוקול). QUIC, לעומתו, כולל הצפנה כמאפיין ליבה. כל הנתונים המועברים באמצעות QUIC מוצפנים כברירת מחדל, מה שמספק אבטחה ופרטיות טובים יותר.

2) מנה לפחות שני הבדלים עיקריים בין Cubic ל-Vegas

שני אלה הם אלגוריתמים לשליטת צפיפות (congestion control) המשמשים בפרוטוקולי רשת כמו TCP. להלן שני הבדלים עיקריים:

מנגנון מבוסס משוב: Cubic משתמש בגודל החלון וב-RTT של החבילות כדי לחשב את קצב הגידול של חלון החבילות. קצב הצמיחה מותאם על

סמך רמת העומס ברשת, אשר מחושבות באמצעות גודל החלונות הנוכחיים והקודמים והזמן שלוקח לשדר חבילות בהם.
לעומת זאת, Vegas משתמש בגישה פרואקטיבית יותר המודדת את זמן הנסיעה הלוך ושוב (RTT) של החבילות כעת כדי להעריך את העומס ברשת ומתאימה את חלון בהתאם למצב העכשווי.

שחזור אובדן חבילות: Cubic משתמש במנגנון התחלה איטית כדי להתאושש במהירות מאובדני חבילות על ידי איפוס גודל חלון הצפיפות לערך קטן והגדלתו הדרגתית. Vegas, לעומתו, מנסה להימנע מאובדן חבילות על ידי שמירה על RTT עקבי, שעוזר לה לזהות ולהפחית עומס לפני שהוא גורם לאובדן חבילות. אם אכן מתרחש אובדן, Vegas משתמשת בגישת שידור חוזר אגרסיבי יותר מ-Cubic כדי לשחזר את החבילות האבודות. היא מניחה שהרשת עמוסה ומפחיתה את קצב השליחה שלה על ידי הקטנת חלון התעבורה (cwnd) שלה בגורם אלפא. אלפא הוא פרמטר שקובע כמה ה-cwnd יורד כאשר מתרחש אובדן מנות. בדרך כלל, אלפא מוגדר ל-1/8.

3) <u>הסבר מהו פרוטוקול BGP, במה הוא שונה מ-OSPF והאם הוא עובד על פי מסלולים קצרים</u>

פרוטוקול הניתוב Border Gateway Protocol (BGP) משמש באינטרנט כדי להעביר נתוני ניתוב בין מערכות אוטונומיות שונות (AS), כל אחת קבוצה של נתבים תחת מנהל טכני יחיד. ספקי שירותי אינטרנט (ISP) משתמשים ב-BGP כדי להעביר פרטי ניתוב ולמצוא את הנתיב הקצר ביותר עבור חבילות נתונים כדי להגיע ליעדם.

BGP נועד לתפקד על פני מספר מערכות אוטונומיות, בניגוד לפרוטוקול Open Shortest Path First (OSPF), המשמש רק בתוך מערכת אוטונומית אחת. לעומת BGP, שהוא פרוטוקול שער חיצוני (EGP) המשמש לגילוי הנתיב הקצר ביותר עבור חבילות נתונים בין רשתות שונות המופעלות על ידי ארגונים שונים, OSPF קובע את הנתיב הקצר ביותר עבור חבילות נתונים בתוך רשת יחידה.
OSPF בונה מפה טופולוגית של הרשת על ידי החלפת הודעות קישור (LSAs) עם נתבים אחרים באותה רשת AS. OSPF מאפשר ניתוב יעיל וממוטב יותר בתוך רשת אחת, שכן הוא לוקח בחשבון מדדים שונים, כגון מהירות קישור ורוחב פס, כדי לחשב את הנתיב הקצר ביותר ליעד.

4) הוסיפו את הנתונים לטבלה הזו על בסיס תהליך ההודעות של הפרויקט שלכם. הסבירו איך ההודעות ישתנו אם יהיה NATבין המשתמש לשרתים והאם תשתמשו בפרוטוקול QUIC

איפה שמופיע x,y,z בטבלה זה מכיוון שאנו מעניקים כתובות IP רנדומליות. כתובת ה-mac היא של כרטיס הרשת ואם נעשית באותו מחשב היא יוצאת זהה בכל השרתים.

Application	Port Src	Port Des	IP Src	IP Des	Mac Src	Mac Des
ftp	30251	20599	10.0.3.X	10.0.3.Y	08:00:27:15:5a:a0	08:00:27:15:5a:a0
client	20599	30251	10.0.3.Y	10.0.3.X	08:00:27:15:5a:a0	08:00:27:15:5a:a0
dhcp	67	68	10.0.3.1	10.0.3.0/24	08:00:27:15:5a:a0	08:00:27:15:5a:a0
dns	53	20599	10.0.3.Z	10.0.3.Y	08:00:27:15:5a:a0	08:00:27:15:5a:a0

תרגום כתובות רשת (NAT) היא טכניקת ניתוב ברשת מחשבים, בה נכתבות מחדש כתובות ה-IP. אם הוא קיים בין משתמש לשרתים, ההודעות ביניהם יושפעו בכמה דרכים:

שינויים בכתובת IP: ה-NAT יחליף את כתובת ה-IP המקורית של ההודעות היוצאות מהמכשיר של המשתמש לכתובת IP של הנתב עצמו. באופן הזה, אנו שומרים על המידע של כתובות IP ברשת הפנימית כחסויים כאשר יגיעו לשרת.

שינויים במספר ה-Port: התקן NAT ישנה גם את מספר הפורט המקורי של ההודעות היוצאות למספר פורט גבוה ייחודי שישמש כאינדקס לטבלת ה-NAT כדי לעקוב אחר החיבורים השונים בין מהמכשיר של המשתמש.

אולם, כתובת ה-MAC של המכשיר של המשתמש לא תשתנה בעת תקשורת עם שרתים דרך התקן NAT.

5) הסבירו את ההבדלים בין פרוטוקול ARP ל-DNS

DNS (מערכת שמות מתחם) ו-ARP (פרוטוקול תרגום כתובות) הן שתיהן טכנולוגיות רשת משמעותיות, אך יש להן מטרות שונות.

ARP משמש למיפוי כתובת פיזית או MAC של מכשיר לכתובת ה-IP שלו ברשת מקומית. (למשל, בפרויקט שלנו הוספנו רשימה בשרת ה-DHCP שעוקבת אחר הכתובות שהוענקו ואיזה כתובת MAC כל אחת שייכת.) למעשה, מכשיר שולח בקשת ARP לגלות את כתובת ה-MAC של מכשיר היעד בכל פעם שהוא צריך לתקשר עם מכשיר אחר באותה רשת. כל המכשירים ברשת מקבלים את בקשת ה-ARP, והמכשיר עם כתובת ה-IP התואמת עונה עם כתובת ה-MAC שלו. לאחר מכן המכשיר שחיפש את הכתובת שומר את הנתונים הללו לשימוש מאוחר יותר במטמון ה-ARP שלו.

DNS, לעומת זאת, משמש למיפוי שמות מתחמים הניתנים לקריאה על ידי אדם לכתובות IP באינטרנט. כאשר משתמש מקליד שם דומיין בדפדפן אינטרנט, הדפדפן שולח בקשת DNS לשרת DNS כדי להגדיר את שם הדומיין לכתובת IP. שרת ה-DNS מחפש את כתובת ה-IP השייכת לשם הדומיין ומחזיר אותה לדפדפן, אשר לאחר מכן משתמש בכתובת ה-IP כדי ליצור חיבור לשרת המארח את האתר.
תהליך הבירור עובר דרך שרתים המכונים "רמת השורש" (root servers). שרתים אלה מסוגלים להפנות את המבקש לשרתים הרלוונטיים עבור כל כתובת. השרתים ברמה הבאה אחראיים על שמות תחום מהרמה העליונה (Top level domains), כלומר על כל שמות התחום אשר משתמשים בסיומת אינטרנט מסוימת. (il או com) והם ימשיכו לנתב אותו לשרתים הבאים בתור לפי כתובת ה-DNS עד שנגיע לשרת שמחזיק ב-IP.

צילומים של unit testing של ה-TCP וה-RUDP כולל DNS ו-DHCP

בפרויקט יצרנו קובץ test_client עבור בדיקה שה-DNS וה- DHCP עובדים כראוי.

```
class TestClient(unittest.TestCase):
    def test_dhcp_dns(self):
        mac = Net_Client.get_mac()
        Net_Client.send_discover(mac)
        pkt = Net_Client.receive_offer()
        server_ip = get_anticipated_server_ip()
        offered_ip = pkt[0][BOOTP].yiaddr
        self.assertEqual(pkt[0][IP].src, server_ip)

        Net_Client.send_request(mac, offered_ip, server_ip)
        print("Waiting for DHCP ACK")
        ack_pkt = Net_Client.receive_acknowledge()
        self.assertEqual(ack_pkt[0][BOOTP].yiaddr, offered_ip)

        dns_ip = pkt[0][DHCP].options[4][1]
        # Send a DNS query for ftpdrive.org
        dns_reply = Net_Client.send_DNS_query(dns_ip, offered_ip, Port)
        print("Received DNS reply.")
        dns_msg = dns.message.from_wire(dns_reply)
        # Net_Client.print_dns_reply(dns_reply) # For printing the DNS reply
        domain_name = dns_msg.answer[0].name.to_text()
        self.assertEqual(domain_name, "ftpdrive\.org.")

if __name__ == '__main__':
    unittest.main()
```

לאחר שנריץ את שרת ה-DHCP ואז ה-DNS, נריץ את test_client שיבדוק שאכן שרת ה-DHCP מקשיב בפורט 67, שולח חבילת offer כראוי ושאנו מצליחים לחלץ את ה-IP שלו שהוא קבוע ולכן ניתן לבדוק אם מתאים לחישוב שנעשה בטסט.
טסט נוסף יבדוק שגם שליחת הבקשה לשימוש ואישור מטעם ה-DHCP אכן עובד.
טסט אחרון מוודא ששרת ה-DNS מחזיר תשובה תקינה ושניתן לחלץ את שם הדומיין של אפליקציית ה-FTP.

```
nitay@ubuntu:~/Desktop/FinelProjectReshatot$ sudo python3 -m unittest test_client.py
[sudo] password for nitay:
.
Sent 1 packets.
Received DHCP offer.
Server IP address:  10.0.3.1
Offered IP address:  10.0.3.38
.
Sent 1 packets.
Waiting for DHCP ACK
Received DNS reply.
.
----------------------------------------------------------------------
Ran 1 test in 4.185s

OK
nitay@ubuntu:~/Desktop/FinelProjectReshatot$
```